Лабораторная работа № 1

Тема: Работа с классом CMatrix

Задание:

1. Реализовать функцию (не член класса **CMatrix**) для вывода переменной типа **CMatrix** (матрицы или вектора) в окно Windows

void PrintMatrix(CDC& dc, int x, int y, CMatrix& M);

// dc – ссылка на контекст
// x, y – координаты точки в окне, откуда начинается вывод
// M – выводимая матрица

2. Тестирование класса CMatrix.

В конструкторе класса CChildView определить матрицы:
A(3x3), B(3x3), V1(3x1) – вектор, V2(3x1) – вектор.
Вычислить:
C1=A+B, C2=A·B, D=A·V1, $q=V1^T \cdot V2$, $p=V1^T \cdot A \cdot V2$.
Матрицы C1, C2, D и числа q, p вывести в окно, используя метод **PrintMatrix (…)**.
Вычисления выполняются при выборе пункта меню **Tests►Matrix**
Результаты вычислений проверить в пакете Mathcad.

3. Реализовать функции (не члены класса **CMatrix**)

CMatrix VectorMult(CMatrix& V1,CMatrix& V2);
// Вычисляет векторное произведение векторов V1 и V2

//----------

double ScalarMult(CMatrix& V1,CMatrix& V2);

// Вычисляет скалярное произведение векторов V1 и V2

//----------

double ModVec(CMatrix& V);

// Вычисляет модуль вектора V

//----------

double CosV1V2(CMatrix& V1,CMatrix& V2);

// Вычисляет КОСИНУС угла между векторами V1 и V2

CMatrix SphereToCart(CMatrix& PView);

// Преобразует сферические координаты PView точки в декартовы

// PView(0) – r – расстояние до точки.

// PView(1) - fi - азимут(отсчет от оси X), град.

// PView(2) - q - угол(отсчетот оси Z), град.

// Результат: R(0)- x, R(1)- y, R(2)- z

Файлы *.h определяются в файле LibGraph.

Файлы *.cpp определяются в файле LibGraph.

Выполнить тестирование разработанных функций с использованием векторов V1 и V2.

Вычисления выполняются при выборе пункта меню **Test►Functions**
Результаты вычислений вывести в окно, используя метод **PrintMatrix (…)**.
Результаты вычислений проверить в пакете Mathcad.

Реализация:

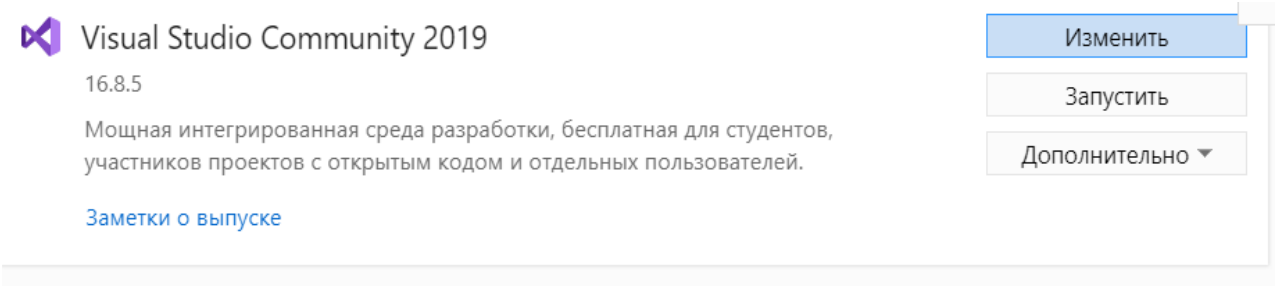
Неободимо реализовать функцию для вывода переменной типа CMatrix в окно Windows, реализовать функции CMatrix VectorMult(CMatrix& V1,CMatrix& V2), double ScalarMult(CMatrix& V1,CMatrix& V2), double ModVec(CMatrix& V), double CosV1V2(CMatrix& V1,CMatrix& V2), CMatrix SphereToCart(CMatrix& PView).

Начало

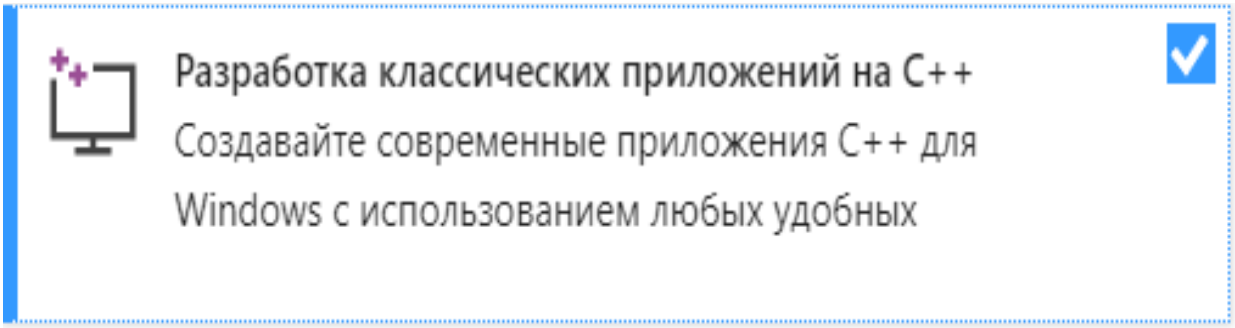
Для выполнения этой лабораторной работой, вам требуется установить библиотеку MFC.

Заходим в Visual Studio Installer. Для добавления новых компонентов нажмите кнопку “Изменить”.

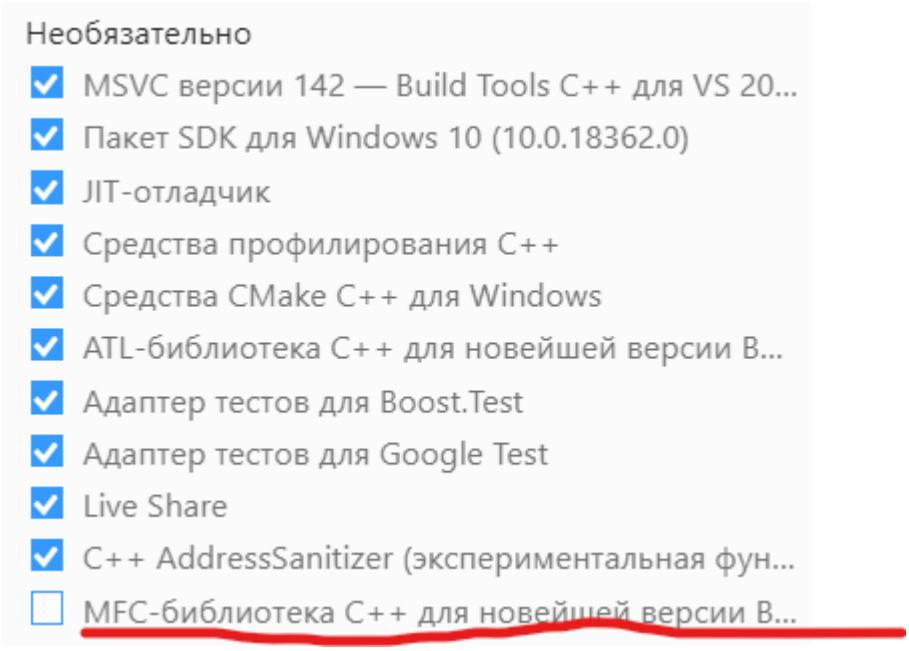
Заходим в Visual Studio Installer. Для добавления новых компонентов нажмите кнопку “Изменить”.



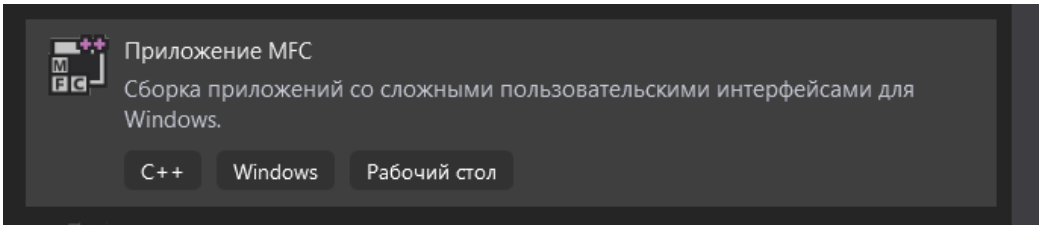
Выбираем пункт Разработка классических приложений на C++.



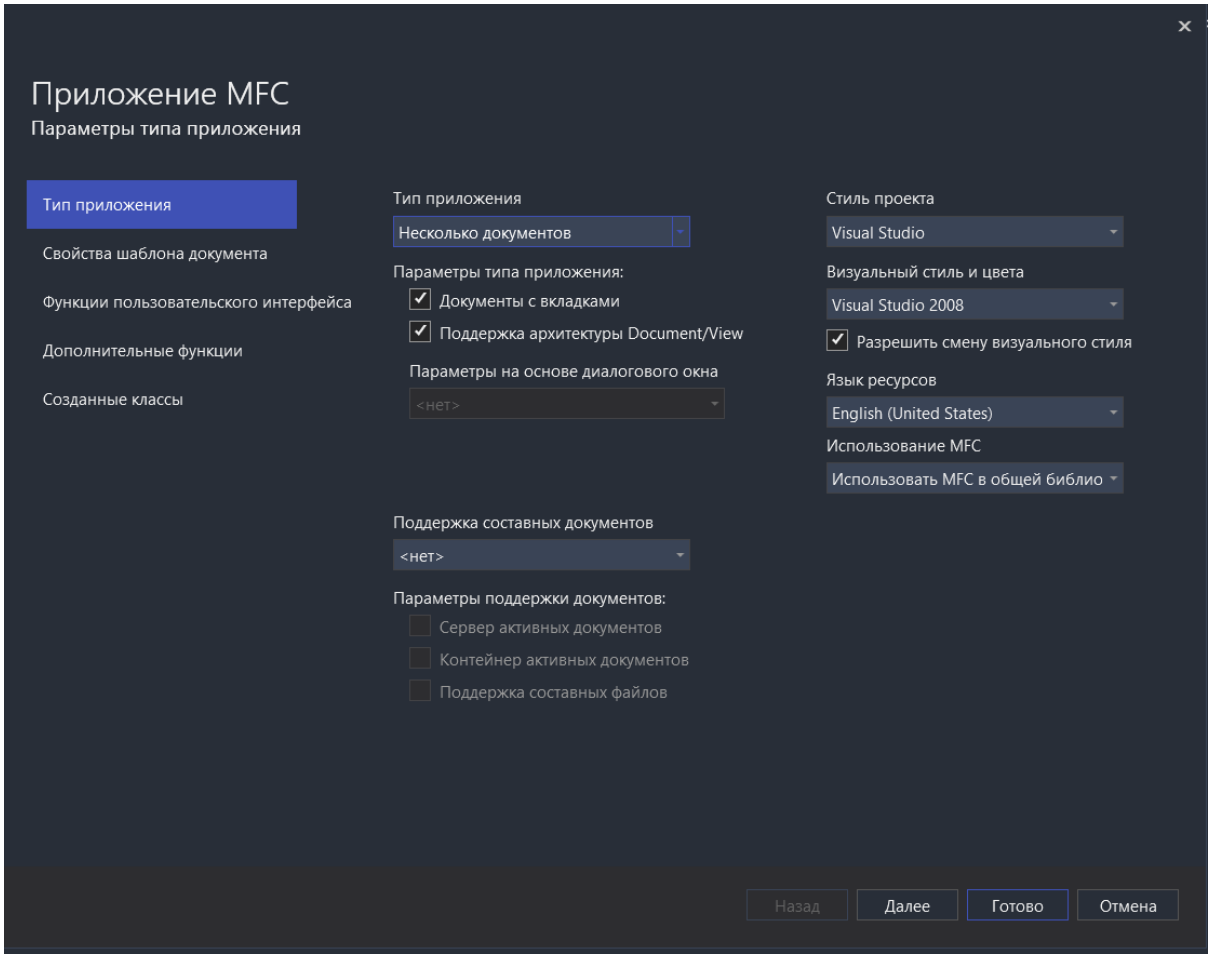
Далее в меню пакетов выбираем MFC-библиотеку C++.

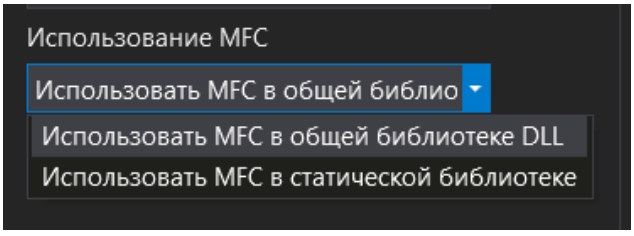


Создаём приложение MFC



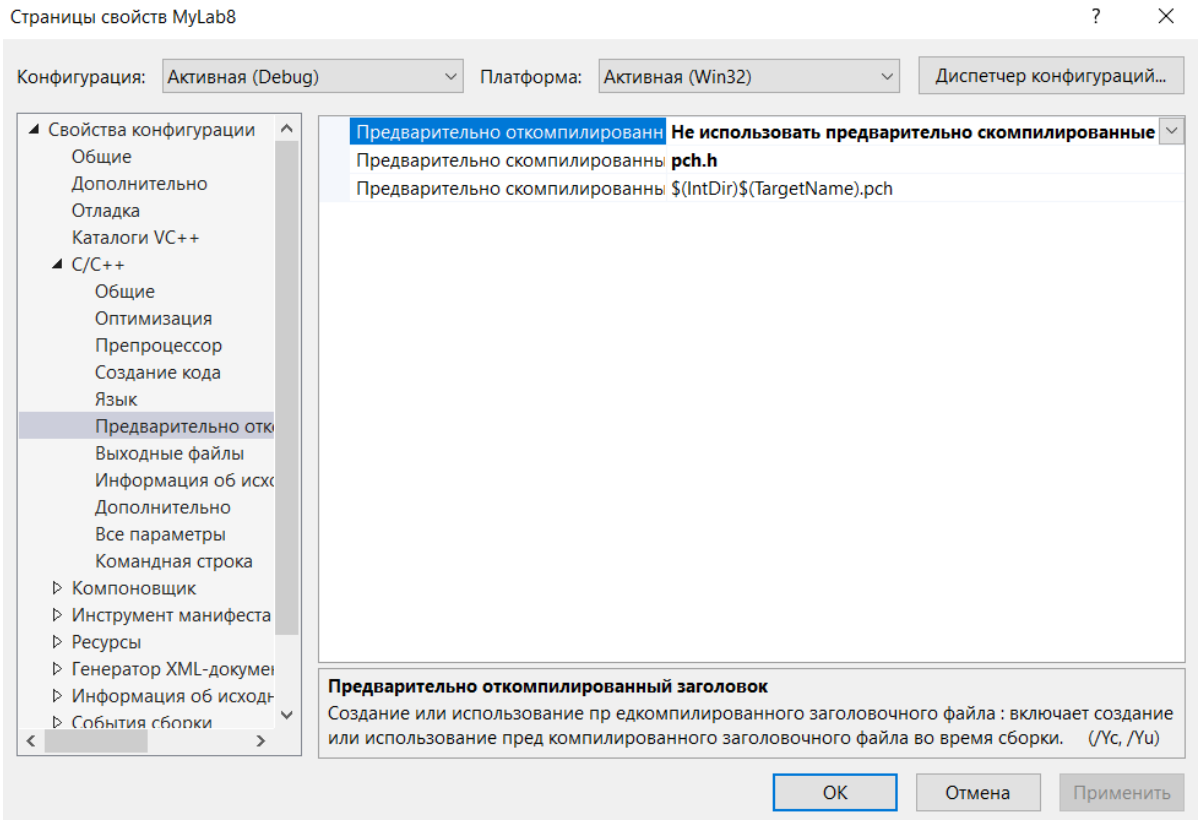
В настройках убеждаемся, что библиотека MFC будет использоваться в общей DLL



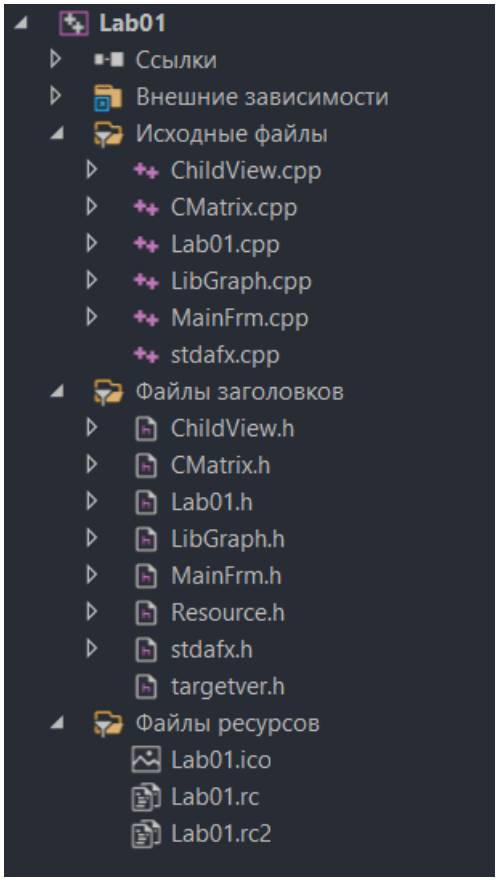


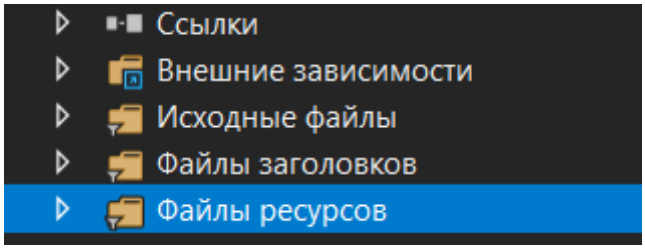
После удаления мусора, заходим в свойства проекта. Тыкаем на C/C++ и выбираем Предварительно. И ставим что мы НЕ БУДЕМ использовать этот pch.h. Мы будем использовать stdafx для мерджина.

Теперь заходим в свойства проекта, C/C++, Предварительно откомпилированные и выбрать “Не использовать предварительно скомпилированные заголовки”.



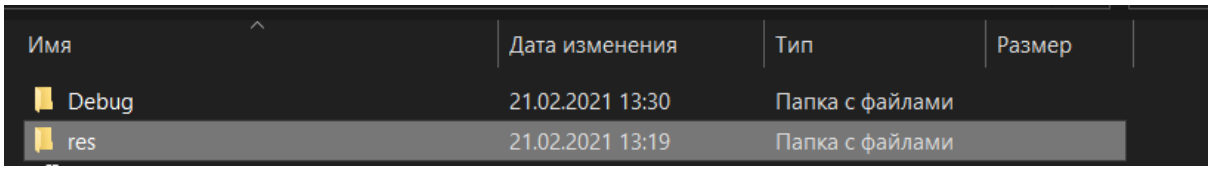
Чтобы начать работу над проектом, требуется удалить все созданные, при создании приложения, файлы.





Нам понадобятся файлы ресурсов.

Эти файлы располагаются в папке с проектом в папке res

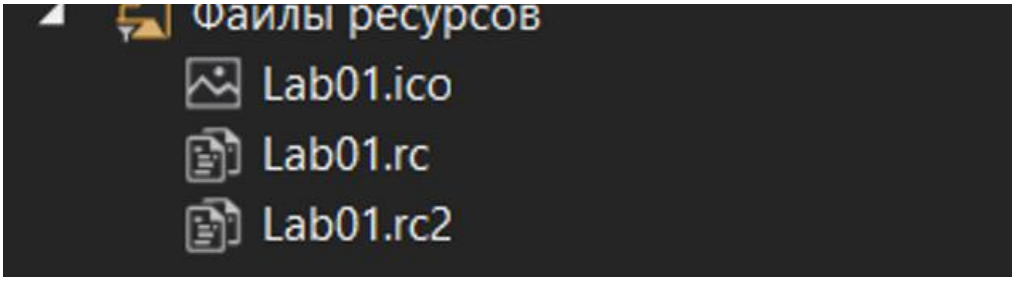


Если этой папки нет, то нужно ее создать. Или попробовать откомпилировать проект.

Изначально, после создания проекта, в ресурсах появится очень много файлов. Их все УДАЛЯЕМ, а не исключаем из проекта.

Вместе с этой лабораторной работой будет архив с нужными ресурсами. Их всего 3. Закидываем эти ресурсы в папку res, и добавляем в наш проект.

Получится вот так.



Их не надо менять, код, написанный далее. Он будет забинджен под эти ресурсы.

Далее создадим файл stdafx.h и stdafx.cpp, в котором будет служебная информация а также подключения нужных нам заголовочных файлов.

stdafx.h:

```
// stdafx.h: включите файл для добавления стандартных системных
файлов
//или конкретных файлов проектов, часто используемых,
// но редко изменяемых

#pragma once

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN            // Исключите редко используемые
компоненты из заголовков Windows
#endif

#include "targetver.h"

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // некоторые
конструкторы CString будут явными

// отключает функцию скрытия некоторых общих и часто пропускаемых
предупреждений MFC
#define _AFX_ALL_WARNINGS
```

```
#include <afxwin.h>         // основные и стандартные компоненты
MFC
#include <afxext.h>         // расширения MFC


#include <afxdisp.h>        // классы автоматизации MFC



#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>           // поддержка MFC для типовых эле-
ментов управления Internet Explorer 4
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // поддержка MFC для типовых эле-
ментов управления Windows
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxcontrolbars.h>     // поддержка MFC для лент и пане-
лей управления

#include "CMatrix.h"//
#include "LibGraph.h"//
#include <vector>//

using std::vector;//


#ifdef _UNICODE
#if defined _M_IX86
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' proces-
sorArchitecture='x86' publicKeyToken='6595b64144ccf1df' lan-
guage='*'\"")
#elif defined _M_X64
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' proces-
sorArchitecture='amd64' publicKeyToken='6595b64144ccf1df' lan-
guage='*'\"")
#else
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' proces-
sorArchitecture='*' publicKeyToken='6595b64144ccf1df' lan-
guage='*'\"")
#endif
#endif
```

Большинство заголовочных файлов будет подчеркнуто красным. Реализацию этих файлов напишем позже.

Теперь напишем stdafx.cpp:

CMatrix.h:

```
#ifndef CMATRIXH
# define CMATRIXH 1
class CMatrix
{
      double **array;
      int n_rows;                                        // Число строк
      int n_cols;                                        // Число столбцов
public:
      CMatrix();                                    // Конструктор по умолчанию (1
на 1)
      CMatrix(int, int);                      // Конструктор
      CMatrix(int);                                  // Конструктор -вектора (один
столбец)
      CMatrix(const CMatrix&);               // Конструктор копирования
      ~CMatrix();
      double &operator()(int, int);         // Выбор элемента матрицы по индексу
      double &operator()(int);              // Выбор элемента вектора по индексу
      CMatrix operator-();                  // Оператор "-"
      CMatrix operator=(const CMatrix&);     // Оператор "Присвоить":    M1=M2
      CMatrix operator*(CMatrix&);         // Оператор "Произведение": M1*M2
      CMatrix operator+(CMatrix&);          // Оператор "+": M1+M2
      CMatrix operator-(CMatrix&);          // Оператор "-": M1-M2
      CMatrix operator+(double);            // Оператор "+": M+a
      CMatrix operator-(double);            // Оператор "-": M-a
      int rows()const { return n_rows; };   // Возвращает число строк
      int cols()const { return n_cols; };    // Возвращает число строк
      CMatrix Transp();                            // Возвращает матри-
цу,транспонированную к текущей
      CMatrix GetRow(int);                  // Возвращает строку по номеру
      CMatrix GetRow(int, int, int);
      CMatrix GetCol(int);                  // Возвращает столбец по номеру
      CMatrix GetCol(int, int, int);
      CMatrix RedimMatrix(int, int);        // Изменяет размер матрицы с уничто-
жением данных
      CMatrix RedimData(int, int);          // Изменяет размер матрицы с сохра-
нением данных,
```

```
                                                        //которые можно сохра-
нить
      CMatrix RedimMatrix(int);           // Изменяет размер матрицы с уничтоже-
нием данных
      CMatrix RedimData(int);             // Изменяет размер матрицы с сохране-
нием данных,
                                                        //которые можно сохра-
нить
      double MaxElement();                  // Максимальный элемент матрицы
      double MinElement();                  // Минимальный элемент матрицы
};


#endif
```

CMatrix.cpp:

```
#include "stdafx.h"
#include "CMatrix.h"

CMatrix::CMatrix()
{
        n_rows = 1;
        n_cols = 1;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}

//---------------------------------------------------------------------------
CMatrix::CMatrix(int Nrow, int Ncol)
// Nrow - число строк
// Ncol - число столбцов
{
        n_rows = Nrow;
        n_cols = Ncol;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}

//---------------------------------------------------------------------------
CMatrix::CMatrix(int Nrow)  //Вектор
// Nrow - число строк
```

```
{
        n_rows = Nrow;
        n_cols = 1;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}
//--------------------------------------------------------------------------
CMatrix::~CMatrix()
{
        for (int i = 0; i < n_rows; i++) delete array[i];
        delete array;
}

//--------------------------------------------------------------------------
double &CMatrix::operator()(int i, int j)
// i - номер строки
// j - номер столбца
{
        if ((i > n_rows - 1) || (j > n_cols - 1))     //  проверка выхода за
диапазон
        {
          TCHAR* error = _T("CMatrix::operator(int,int): выход индекса за гра-
ницу диапазона ");
          MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
          exit(1);
        }
        return array[i][j];
}

//--------------------------------------------------------------------------
double &CMatrix::operator()(int i)
// i - номер строки для вектора
{
        if (n_cols > 1)     //  Число столбцов больше одного
        {
          wchar_t* error = L"CMatrix::operator(int): объект не вектор - число
столбцов больше 1 ";
          MessageBox(NULL, error, L"Ошибка", MB_ICONSTOP);
          exit(1);
        }
        if (i > n_rows - 1)     //  проверка выхода за диапазон
        {
          TCHAR* error = TEXT("CMatrix::operator(int): выход индекса за грани-
цу диапазона ");
          MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
```

```
          exit(1);
        }
        return array[i][0];
}
//---------------------------------------------------------------------------
CMatrix CMatrix::operator-()
// Оператор -M
{
        CMatrix Temp(n_rows, n_cols);
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) Temp(i, j) = -array[i][j];
        return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator+(CMatrix& M)
// Оператор M1+M2
{
        int bb = (n_rows == M.rows()) && (n_cols == M.cols());
        if (!bb)
        {
          wchar_t* error = L"CMatrix::operator(+): несоответствие размерностей
матриц ";
          MessageBox(NULL, error, L"Ошибка", MB_ICONSTOP);
          exit(1);
        }
        CMatrix Temp(*this);
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) Temp(i, j) += M(i, j);
        return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator-(CMatrix& M)
// Оператор M1-M2
{
        int bb = (n_rows == M.rows()) && (n_cols == M.cols());
        if (!bb)
        {
          wchar_t* error = L"CMatrix::operator(-): несоответствие размерностей
матриц ";
          MessageBox(NULL, error, L"Ошибка", MB_ICONSTOP);
          exit(1);
        }
        CMatrix Temp(*this);
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) Temp(i, j) -= M(i, j);
```

```
        return Temp;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::operator*(CMatrix& M)
// Умножение на матрицу M
{
        double sum;
        int nn = M.rows();
        int mm = M.cols();
        CMatrix Temp(n_rows, mm);
        if (n_cols == nn)
        {
          for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < mm; j++)
            {
              sum = 0;
              for (int k = 0; k < n_cols; k++) sum += (*this)(i, k)*M(k, j);
              Temp(i, j) = sum;
            }
        }
        else
        {
          TCHAR* error = TEXT("CMatrix::operator*: несоответствие размерностей
матриц ");
          MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
          exit(1);
        }
        return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator=(const CMatrix& M)
// Оператор присваивания M1=M
{
        if (this == &M) return *this;
        int nn = M.rows();
        int mm = M.cols();
        if ((n_rows == nn) && (n_cols == mm))
        {
          for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++) array[i][j] = M.array[i][j];
        }
        else   // для ошибки размерностей
        {
          TCHAR* error = TEXT("CMatrix::operator=: несоответствие размерностей
матриц");
          MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
```

```
          exit(1);
        }
        return *this;
}

//---------------------------------------------------------------------------
CMatrix::CMatrix(const CMatrix &M) // Конструктор копирования
{
        n_rows = M.n_rows;
        n_cols = M.n_cols;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) array[i][j] = M.array[i][j];
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator+(double x)
// Оператор M+x, где M - матрица, x - число
{
        CMatrix Temp(*this);
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) Temp(i, j) += x;
        return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator-(double x)
// Оператор M+x, где M - матрица, x - число
{
        CMatrix Temp(*this);
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) Temp(i, j) -= x;
        return Temp;
}



//---------------------------------------------------------------------------
CMatrix CMatrix::Transp()
// Возвращает матрицу,транспонированную к (*this)
{
        CMatrix Temp(n_cols, n_rows);
        for (int i = 0; i < n_cols; i++)
          for (int j = 0; j < n_rows; j++) Temp(i, j) = array[j][i];
        return Temp;
}
```

```
//---------------------------------------------------------------------------
CMatrix CMatrix::GetRow(int k)
// Возвращает строку матрицы по номеру k
{
        if (k > n_rows - 1)
        {
          wchar_t* error = L"CMatrix::GetRow(int k): параметр k превышает чис-
ло строк ";
          MessageBox(NULL, error, L"Ошибка", MB_ICONSTOP);
          exit(1);
        }
        CMatrix M(1, n_cols);
        for (int i = 0; i < n_cols; i++)M(0, i) = (*this)(k, i);
        return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::GetRow(int k, int n, int m)
// Возвращает подстроку из строки матрицы с номером k
// n - номер первого элемента в строке
// m - номер последнего элемента в строке
{
        int b1 = (k >= 0) && (k < n_rows);
        int b2 = (n >= 0) && (n <= m);
        int b3 = (m >= 0) && (m < n_cols);
        int b4 = b1 && b2&&b3;
        if (!b4)
        {
          wchar_t* error = L"CMatrix::GetRow(int k,int n, int m):ошибка в
параметрах ";
          MessageBox(NULL, error, L"Ошибка", MB_ICONSTOP);
          exit(1);
        }
        int nCols = m - n + 1;
        CMatrix M(1, nCols);
        for (int i = n; i <= m; i++)M(0, i - n) = (*this)(k, i);
        return M;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::GetCol(int k)
// Возвращает столбец матрицы по номеру k
{
        if (k > n_cols - 1)
        {
          wchar_t* error = L"CMatrix::GetCol(int k): параметр k превышает чис-
ло столбцов ";
```

```
          MessageBox(NULL, error, L"Ошибка", MB_ICONSTOP);
          exit(1);
        }
        CMatrix M(n_rows, 1);
        for (int i = 0; i < n_rows; i++)M(i, 0) = (*this)(i, k);
        return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::GetCol(int k, int n, int m)
// Возвращает подстолбец из столбца матрицы с номером k
// n - номер первого элемента в столбце
// m - номер последнего элемента в столбце
{
        int b1 = (k >= 0) && (k < n_cols);
        int b2 = (n >= 0) && (n <= m);
        int b3 = (m >= 0) && (m < n_rows);
        int b4 = b1 && b2&&b3;
        if (!b4)
        {
          wchar_t* error = L"CMatrix::GetCol(int k,int n, int m):ошибка в
параметрах ";
          MessageBox(NULL, error, L"Ошибка", MB_ICONSTOP);
          exit(1);
        }
        int nRows = m - n + 1;
        CMatrix M(nRows, 1);
        for (int i = n; i <= m; i++)M(i - n, 0) = (*this)(i, k);
        return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::RedimMatrix(int NewRow, int NewCol)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol - новое число столбцов
{
        for (int i = 0; i < n_rows; i++) delete array[i];
        delete array;
        n_rows = NewRow;
        n_cols = NewCol;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) array[i][j] = 0;
        return (*this);
}

//---------------------------------------------------------------------------
```

```
CMatrix CMatrix::RedimData(int NewRow, int NewCol)
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol - новое число столбцов
{
        CMatrix Temp = (*this);
        this->RedimMatrix(NewRow, NewCol);
        int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() :
(*this).rows();
        int min_cols = Temp.cols() < (*this).cols() ? Temp.cols() :
(*this).cols();
        for (int i = 0; i < min_rows; i++)
          for (int j = 0; j < min_cols; j++) (*this)(i, j) = Temp(i, j);
        return (*this);
}


//---------------------------------------------------------------------------
CMatrix CMatrix::RedimMatrix(int NewRow)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol=1
{
        for (int i = 0; i < n_rows; i++) delete array[i];
        delete array;
        n_rows = NewRow;
        n_cols = 1;
        array = new double*[n_rows];
        for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
        for (int i = 0; i < n_rows; i++)
          for (int j = 0; j < n_cols; j++) array[i][j] = 0;
        return (*this);
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimData(int NewRow)
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol=1
{
        CMatrix Temp = (*this);
        this->RedimMatrix(NewRow);
        int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() :
(*this).rows();
        for (int i = 0; i < min_rows; i++)(*this)(i) = Temp(i);
        return (*this);
}
```

```
//---------------------------------------------------------------------------
double CMatrix::MaxElement()
// Максимальное значение элементов матрицы
{
        double max = (*this)(0, 0);
        for (int i = 0; i < (this->rows()); i++)
          for (int j = 0; j < (this->cols()); j++) if ((*this)(i, j) > max)
max = (*this)(i, j);
        return max;
}

//---------------------------------------------------------------------------
double CMatrix::MinElement()
// Минимальное значение элементов матрицы
{
        double min = (*this)(0, 0);
        for (int i = 0; i < (this->rows()); i++)
          for (int j = 0; j < (this->cols()); j++) if ((*this)(i, j) < min)
min = (*this)(i, j);
        return min;
}
```

Добавим MainFrm, это компонент класса CMainFrame
MainFrm.h:

```
// MainFrm.h: интерфейс класса CMainFrame
//

#pragma once
#include "ChildView.h"

class CMainFrame : public CFrameWnd
{

public:
     CMainFrame() noexcept;
protected:
     DECLARE_DYNAMIC(CMainFrame)

// Атрибуты
public:
```

```
// Операции
public:

// Переопределение
public:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo);

// Реализация
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // встроенные члены панели элементов управления
	CStatusBar        m_wndStatusBar;
	CChildView    m_wndView;

// Созданные функции схемы сообщений
protected:
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnSetFocus(CWnd *pOldWnd);
	DECLARE_MESSAGE_MAP()
};
```

MainFrm.cpp:

```
// MainFrm.cpp: реализация класса CMainFrame
//

#include "stdafx.h"
#include "Lab01.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)
```

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	ON_WM_CREATE()
	ON_WM_SETFOCUS()
END_MESSAGE_MAP()

static UINT indicators[] =
{
	ID_SEPARATOR,           // индикатор строки состояния
	ID_INDICATOR_CAPS,
	ID_INDICATOR_NUM,
	ID_INDICATOR_SCRL,
};

// Создание или уничтожение CMainFrame

CMainFrame::CMainFrame() noexcept
{
	// TODO: добавьте код инициализации члена
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
		return -1;

	// создать представление для размещения рабочей области рамки
	if (!m_wndView.Create(nullptr, nullptr, AFX_WS_DEFAULT_VIEW, CRect(0, 0, 0,
0), this, AFX_IDW_PANE_FIRST, nullptr))
	{
		TRACE0("Не удалось создать окно представлений\n");
		return -1;
	}

	if (!m_wndStatusBar.Create(this))
	{
		TRACE0("Не удалось создать строку состояния\n");
		return -1;      // не удалось создать
	}
	m_wndStatusBar.SetIndicators(indicators, sizeof(indicators)/sizeof(UINT));

	return 0;
}
```

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	if( !CFrameWnd::PreCreateWindow(cs) )
		return FALSE;
	// TODO: изменить класс Window или стили посредством изменения
	//  CREATESTRUCT cs

	cs.style = WS_OVERLAPPED | WS_CAPTION | FWS_ADDTOTITLE
		| WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX;

	cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
	cs.lpszClass = AfxRegisterWndClass(0);
	return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CFrameWnd::Dump(dc);
}
#endif //_DEBUG


// Обработчики сообщений CMainFrame

void CMainFrame::OnSetFocus(CWnd* /*pOldWnd*/)
{
	// передача фокуса окну представления
	m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo)
{
	// разрешить ошибки в представлении при выполнении команды
	if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
		return TRUE;

	// в противном случае выполняется обработка по умолчанию
```

```
	return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
```

ChildView.h:

```
// ChildView.h: интерфейс класса CChildView
//


#pragma once


// Окно CChildView

class CChildView : public CWnd
{
// Создание
public:
	CChildView();

	vector<CMatrix> arr;                    // создаем вектор
	ControllerCMatrix controller;           // и контроллер (класс для операций с
матрицами)
// Атрибуты
public:

// Операции
public:

// Переопределение
	protected:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Реализация
public:
	virtual ~CChildView();

	// Созданные функции схемы сообщений
protected:
	afx_msg void OnPaint();
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnTestMatrix();                // тестирование матриц
```

```
	afx_msg void OnTestFunctions();			// функций
};
```

ChildView.cpp:

```
// ChildView.cpp: реализация класса CChildView
//

#include "stdafx.h"
#include "Lab01.h"
#include "ChildView.h"

#ifdef _DEBUG
//#define new DEBUG_NEW
#endif


// CChildView

CChildView::CChildView()
{                                                                   // конструктор
	CMatrix A(3, 3), B(3, 3), V1(3), V2(3);                             // создание
матриц и векторов
	controller.InitMatrix(A); controller.InitMatrix(B);             // и их
инициализация
	controller.InitMatrix(V1); controller.InitMatrix(V2);
	arr.push_back(A); arr.push_back(B); arr.push_back(V1); // добавление матриц в
конец вектора
	arr.push_back(V2);
}

CChildView::~CChildView()
{
}

// Реализация карты сообщений
BEGIN_MESSAGE_MAP(CChildView, CWnd)
	ON_WM_PAINT()
	ON_COMMAND(ID_TEST_MATRIX, &CChildView::OnTestMatrix)               // команды
(id, функция передачи сообщения,
	ON_COMMAND(ID_TEST_FUNCTIONS, &CChildView::OnTestFunctions)         // которой
сопоставляется команда)
END_MESSAGE_MAP()
```

```
// Обработчики сообщений CChildView

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
	if (!CWnd::PreCreateWindow(cs))
		return FALSE;

	cs.dwExStyle |= WS_EX_CLIENTEDGE;
	cs.style &= ~WS_BORDER;
	cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
		::LoadCursor(nullptr, IDC_ARROW), reinterp-
ret_cast<HBRUSH>(COLOR_WINDOW+1), nullptr);

	return TRUE;
}

void CChildView::OnPaint()
{
	CPaintDC dc(this); // контекст устройства для рисования

	// TODO: Добавьте код обработки сообщений

	// Не вызывайте CWnd::OnPaint() для сообщений рисования
}

                                                        // тестирование матриц
void CChildView::OnTestMatrix()
{
	CClientDC dc(this);                                         // получе-
ние контекста окна
	InvalidateRect(0);
	UpdateWindow();                                             // обновле-
ние окна
	CPen MyPen(PS_DASHDOT, 1, RGB(133, 255, 100));          // создание пера(стиль
пера, ширина пера, цвет пера)
	dc.TextOut(10, 5, _T("Исходные матрицы:"));             // вывод в клиентскую
область (координаты, текст)
	dc.TextOut(10, 70, _T("A = "));
	controller.PrintMatrix(dc, 40, 30, arr[0]);             // функция вывода
матриц
	dc.TextOut(10, 240, _T("B = "));
	controller.PrintMatrix(dc, 40, 200, arr[1]);
	dc.TextOut(250, 5, _T("Исходные векторы:"));
	dc.TextOut(250, 70, _T("V1 = "));
```

```
	controller.PrintMatrix(dc, 285, 30, arr[2]);
	dc.TextOut(330, 70, _T("V2 = "));
	controller.PrintMatrix(dc, 365, 30, arr[3]);

	dc.TextOut(500, 5, _T("Результат:"));
	CMatrix C1 = arr[0] + arr[1];
	dc.TextOut(500, 30, _T("C1 = A + B"));
	controller.PrintMatrix(dc, 500, 50, C1);

	CMatrix C2 = arr[0] * arr[1];
	dc.TextOut(700, 30, _T("C2 = A * B"));
	controller.PrintMatrix(dc, 700, 50, C2);

	CMatrix D = arr[0] * arr[2];
	dc.TextOut(900, 30, _T("D = A * V1"));
	controller.PrintMatrix(dc, 900, 50, D);

	CMatrix q = arr[2].Transp() * arr[3];
	dc.TextOut(500, 200, _T("q = V1^T * V2"));
	controller.PrintMatrix(dc, 500, 250, q);

	CMatrix p = arr[2].Transp() * arr[0] * arr[3];
	dc.TextOut(700, 200, _T("p = V1^T * A * V2"));
	controller.PrintMatrix(dc, 700, 250, p);
}

                                                        // тестирование
функций
void CChildView::OnTestFunctions()
{
	CClientDC dc(this);
	InvalidateRect(0);
	UpdateWindow();                                     // обновление окна
	dc.TextOut(10, 5, _T("Исходные векторы:"));
	dc.TextOut(10, 70, _T("V1 = "));
	controller.PrintMatrix(dc, 45, 30, arr[2]);
	dc.TextOut(85, 70, _T("V2 = "));
	controller.PrintMatrix(dc, 120, 30, arr[3]);

	dc.TextOut(200, 5, _T("Векторное произведение:"));
	auto vec = controller.VectorMult(arr[2], arr[3]);
	controller.PrintMatrix(dc, 200, 30, vec);

	dc.TextOut(420, 5, _T("Скалярное произведение:"));
	auto stringScal = controller.DoubleToString(controller.ScalarMult(arr[2],
arr[3]));
	dc.TextOut(420, 30, stringScal);
```

```
	dc.TextOut(420, 70, _T("Модуль вектора V1:"));
	auto stringMod = controller.DoubleToString(controller.ModVec(arr[2]));
	dc.TextOut(420, 95, stringMod);

	dc.TextOut(420, 135, _T("Косинус между V1 и V2:"));
	auto stringCos = controller.DoubleToString(controller.CosV1V2(arr[2],
arr[3]));
	dc.TextOut(420, 160, stringCos);

	dc.TextOut(10, 200, _T("Преобразует сферические координаты PView точки в
декартовы:"));
	auto v = CMatrix(3);
	controller.InitMatrix(v);
	dc.TextOut(10, 265, _T("PView = "));
	controller.PrintMatrix(dc, 70, 225, v);
	controller.PrintMatrix(dc, 300, 225, controller.SphereToCart(v));

}
```

Lab01.h – основной файл для приложения

```
// Lab01.h: основной файл заголовка для приложения Lab01
//
#pragma once

#ifndef __AFXWIN_H__
	#error "включить stdafx.h до включения этого файла в PCH"
#endif

#include "resource.h"		// основные символы


// CLab01App:
// Сведения о реализации этого класса: Lab01.cpp
//

class CLab01App : public CWinApp
{
public:
	CLab01App() noexcept;


// Переопределение
public:
```

```
	virtual BOOL InitInstance();
	virtual int ExitInstance();

// Реализация

public:
	afx_msg void OnAppAbout();
	DECLARE_MESSAGE_MAP()
};

extern CLab01App theApp;
```

Lab01.cpp

```
// Lab01.cpp: определяет поведение классов для приложения.
//

#include "stdafx.h"
#include "afxwinappex.h"
#include "afxdialogex.h"
#include "Lab01.h"
#include "MainFrm.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CLab01App

BEGIN_MESSAGE_MAP(CLab01App, CWinApp)
	ON_COMMAND(ID_APP_ABOUT, &CLab01App::OnAppAbout)
END_MESSAGE_MAP()


// Создание CLab01App

CLab01App::CLab01App() noexcept
{
	// поддержка диспетчера перезагрузки
	m_dwRestartManagerSupportFlags = AFX_RESTART_MANAGER_SUPPORT_ALL_ASPECTS;
#ifdef _MANAGED
```

```
	// Если приложение построено с поддержкой среды Common Language Runtime (/clr):
	//     1) Этот дополнительный параметр требуется для правильной поддержки работы диспетчера перезагрузки.
	//   2) В своем проекте для сборки необходимо добавить ссылку на System.Windows.Forms.
	System::Windows::Forms::Application::SetUnhandledExceptionMode(System::Windows::Forms::UnhandledExceptionMode::ThrowException);
#endif

	// TODO: замените ниже строку идентификатора приложения строкой уникального идентификатора; рекомендуемый
	// формат для строки: ИмяКомпании.ИмяПродукта.СубПродукт.СведенияОВерсии
	SetAppID(_T("Lab01.AppID.NoVersion"));

	// TODO: добавьте код создания,
	// Размещает весь важный код инициализации в InitInstance
}

// Единственный объект CLab01App

CLab01App theApp;


// Инициализация CLab01App

BOOL CLab01App::InitInstance()
{
	// InitCommonControlsEx() требуются для Windows XP, если манифест
	// приложения использует ComCtl32.dll версии 6 или более поздней версии для включения
	// стилей отображения.  В противном случае будет возникать сбой при создании любого окна.
	INITCOMMONCONTROLSEX InitCtrls;
	InitCtrls.dwSize = sizeof(InitCtrls);
	// Выберите этот параметр для включения всех общих классов управления, которые необходимо использовать
	// в вашем приложении.
	InitCtrls.dwICC = ICC_WIN95_CLASSES;
	InitCommonControlsEx(&InitCtrls);

	CWinApp::InitInstance();


	// Инициализация библиотек OLE
	if (!AfxOleInit())
	{
		AfxMessageBox(IDP_OLE_INIT_FAILED);
```

```
		return FALSE;
	}

	AfxEnableControlContainer();

	EnableTaskbarInteraction(FALSE);

	// Для использования элемента управления RichEdit требуется метод AfxInitRichE-
dit2()
	// AfxInitRichEdit2();

	// Стандартная инициализация
	// Если эти возможности не используются и необходимо уменьшить размер
	// конечного исполняемого файла, необходимо удалить из следующего
	// конкретные процедуры инициализации, которые не требуются
	// Измените раздел реестра, в котором хранятся параметры
	// TODO: следует изменить эту строку на что-нибудь подходящее,
	// например на название организации
	SetRegistryKey(_T("Локальные приложения, созданные с помощью мастера приложе-
ний"));


	// Чтобы создать главное окно, этот код создает новый объект окна
	// рамки, а затем задает его как объект основного окна приложения
	CFrameWnd* pFrame = new CMainFrame;
	if (!pFrame)
		return FALSE;
	m_pMainWnd = pFrame;
	// создайте и загрузите рамку с его ресурсами
	pFrame->LoadFrame(IDR_MAINFRAME,
		WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, nullptr,
		nullptr);


	// Разрешить использование расширенных символов в горячих клавишах меню
	CMFCToolBar::m_bExtCharTranslation = TRUE;




	// Одно и только одно окно было инициализировано, поэтому отобразите и обновите
его
	pFrame->ShowWindow(SW_SHOW);
	pFrame->UpdateWindow();
	return TRUE;
}

int CLab01App::ExitInstance()
```

```
{
	//TODO: обработайте дополнительные ресурсы, которые могли быть добавлены
	AfxOleTerm(FALSE);

	return CWinApp::ExitInstance();
}

// Обработчики сообщений CLab01App


// Диалоговое окно CAboutDlg используется для описания сведений о приложении

class CAboutDlg : public CDialogEx
{
public:
	CAboutDlg() noexcept;

// Данные диалогового окна
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_ABOUTBOX };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // поддержка DDX/DDV

// Реализация
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() noexcept : CDialogEx(IDD_ABOUTBOX)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// Команда приложения для запуска диалога
void CLab01App::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
```

```
// Обработчики сообщений CLab01App
```

Файл с ресурсами: **Resource.h**

```
//{{NO_DEPENDENCIES}}
// Включаемый файл, созданный в Microsoft Visual C++.
// Используется в Lab01.rc
//
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDR_MAINFRAME                   128
#define IDR_Lab01TYPE                   130
#define IDD_DIALOG1                    310
#define ID_TEST_MATRIX                 32771        // определение id для матриц
#define ID_TEST_FUNCTIONS              32772        // и функций

// Следующие стандартные значения для новых объектов
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        312
#define _APS_NEXT_COMMAND_VALUE         32773
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           310
#endif
#endif
```

targetver.h:

```
#pragma once

// Включение SDKDDKVer.h обеспечивает определение самой последней доступной плат-
формы Windows.

// Если требуется выполнить сборку приложения для предыдущей версии Windows, вклю-
чите WinSDKVer.h и
```

```
// задайте для макроопределения _WIN32_WINNT значение поддерживаемой платформы пе-
ред вхождением SDKDDKVer.h.

#include <SDKDDKVer.h>
```

Задание 1.

1. Реализовать функцию (не член класса **CMatrix**) для вывода переменной типа **CMatrix** (матрицы или вектора) в окно Windows

void PrintMatrix(CDC& dc, int x, int y, CMatrix& M);

// dc – ссылка на контекст

// x, y – координаты точки в окне, откуда начинается вывод

// M – выводимая матрица

```
#pragma region Controller
void ControllerCMatrix::PrintMatrix(CDC & dc, int x, int y, CMatrix & M)
{
    for (size_t i = 0; i < M.cols(); i++)
    {
        for (size_t j = 0; j < M.rows(); j++)
        {
            dc.TextOutW(x + i * 40, y + j * 40, DoubleToString(M(j, i)));
        }
    }
}
```

Данная функция выполняет вывод матрицы или вектора в определенный контекст. Контекст передаем первым параметром, координаты точки в окне, откуда начинается ввод вторым параметром, а выводимую матрицу третьим параметром.

```
void ControllerCMatrix::PrintMatrix(CDC & dc, int x, int y, CMatrix & M)

{

for (size_t i = 0; i < M.cols(); i++)

{
```

```
for (size_t j = 0; j < M.rows(); j++)
{
dc.TextOutW(x + i * 40, y + j * 40, DoubleToString(M(j, i)));
}
}
}
```

Задание 2. Тестирование класса CMatrix.

В конструкторе класса CChildView определить матрицы:

A(3x3), B(3x3), V1(3x1) – вектор, V2(3x1) – вектор.

Вычислить:

C1=A+B, C2=A×B, D=A×V1, q=$V1^T$×V2, p=$V1^T$×A×V2.

Матрицы C1, C2, D и числа q, p вывести в окно, используя метод **PrintMatrix (…)**.

Вычисления выполняются при выборе пункта меню **Tests►Matrix**

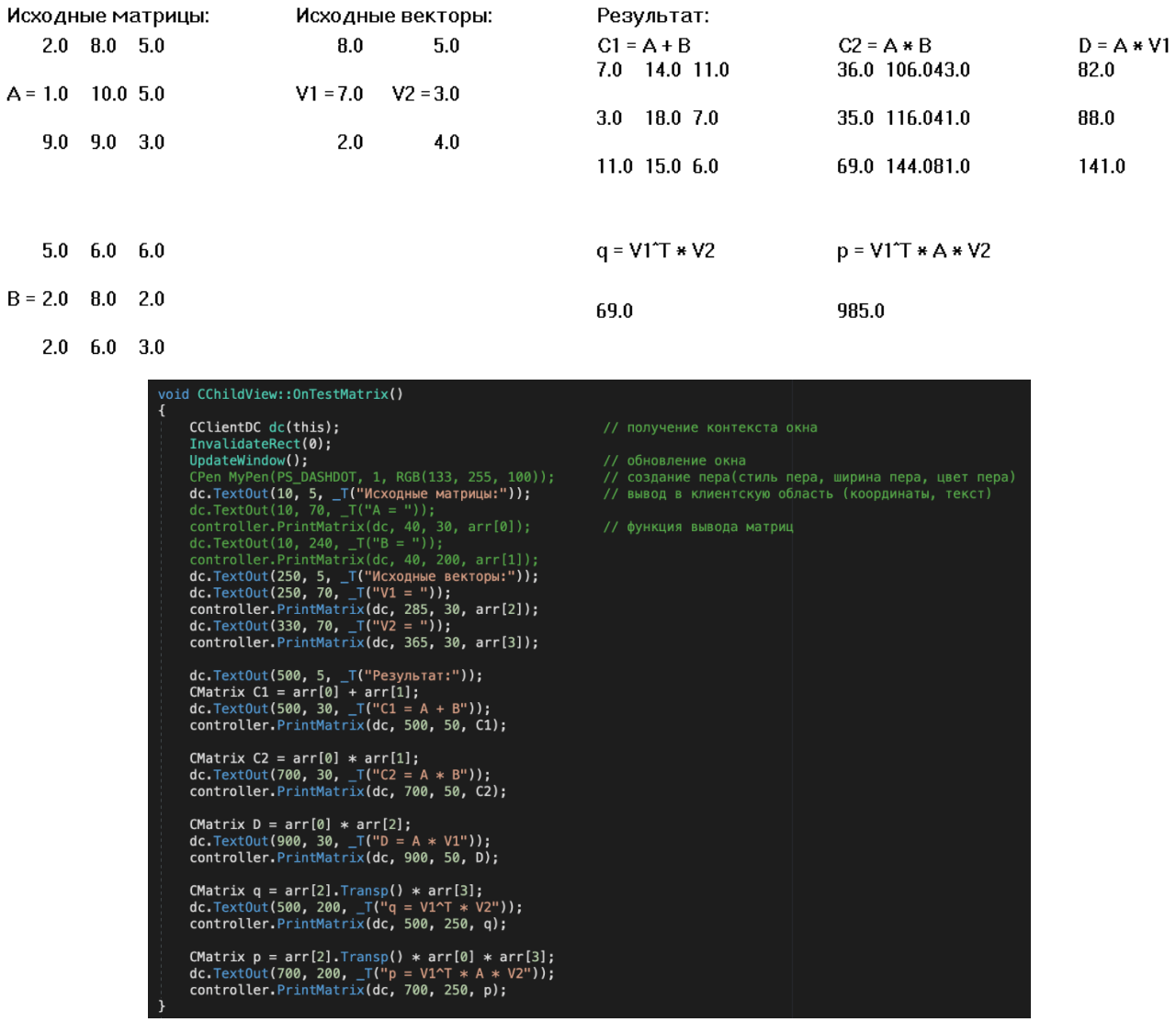
Результаты вычислений проверить в пакете Mathcad

```
CChildView::CChildView()
{
CMatrix A(3, 3), B(3, 3), V1(3), V2(3);
```

```
controller.InitMatrix(A);

controller.InitMatrix(B);

controller.InitMatrix(V1);

controller.InitMatrix(V2);

arr.push_back(A);

arr.push_back(B);

arr.push_back(V1);

arr.push_back(V2);

}
```

Реализация конструктора. Здесь мы создаем матрицы, с помощью controller их инициализируем и добавляем их в конец вектора arr.

Для тестирования вызовем функцию:




```
void CChildView::OnTestMatrix()
{
    CClientDC dc(this);                                 // получение контекста окна
    InvalidateRect(0);
    UpdateWindow();                                     // обновление окна
    CPen MyPen(PS_DASHDOT, 1, RGB(133, 255, 100));      // создание пера(стиль пера, ширина пера, цвет пера)
    dc.TextOut(10, 5, _T("Исходные матрицы:"));         // вывод в клиентскую область (координаты, текст)
    dc.TextOut(10, 70, _T("A = "));
    controller.PrintMatrix(dc, 40, 30, arr[0]);         // функция вывода матриц
    dc.TextOut(10, 240, _T("B = "));
    controller.PrintMatrix(dc, 40, 200, arr[1]);
    dc.TextOut(250, 5, _T("Исходные векторы:"));
    dc.TextOut(250, 70, _T("V1 = "));
    controller.PrintMatrix(dc, 285, 30, arr[2]);
    dc.TextOut(330, 70, _T("V2 = "));
    controller.PrintMatrix(dc, 365, 30, arr[3]);

    dc.TextOut(500, 5, _T("Результат:"));
    CMatrix C1 = arr[0] + arr[1];
    dc.TextOut(500, 30, _T("C1 = A + B"));
    controller.PrintMatrix(dc, 500, 50, C1);

    CMatrix C2 = arr[0] * arr[1];
    dc.TextOut(700, 30, _T("C2 = A * B"));
    controller.PrintMatrix(dc, 700, 50, C2);

    CMatrix D = arr[0] * arr[2];
    dc.TextOut(900, 30, _T("D = A * V1"));
    controller.PrintMatrix(dc, 900, 50, D);

    CMatrix q = arr[2].Transp() * arr[3];
    dc.TextOut(500, 200, _T("q = V1^T * V2"));
    controller.PrintMatrix(dc, 500, 250, q);

    CMatrix p = arr[2].Transp() * arr[0] * arr[3];
    dc.TextOut(700, 200, _T("p = V1^T * A * V2"));
    controller.PrintMatrix(dc, 700, 250, p);
}
```

```
void CChildView::OnTestMatrix()

{

CClientDC dc(this);                                                 // по-
лучение контекста окна

InvalidateRect(0);

UpdateWindow();                                                     // обновление
окна

CPen MyPen(PS_DASHDOT, 1, RGB(133, 255, 100));          // создание пе-
ра(стиль пера, ширина пера, цвет пера)

dc.TextOut(10, 5, _T("Исходные матрицы:"));             // вывод в клиент-
скую область (координаты, текст)
```

```
dc.TextOut(10, 70, _T("A = "));

controller.PrintMatrix(dc, 40, 30, arr[0]);            // функция вывода
матриц

dc.TextOut(10, 240, _T("B = "));

controller.PrintMatrix(dc, 40, 200, arr[1]);

dc.TextOut(250, 5, _T("Исходные векторы:"));

dc.TextOut(250, 70, _T("V1 = "));

controller.PrintMatrix(dc, 285, 30, arr[2]);

dc.TextOut(330, 70, _T("V2 = "));

controller.PrintMatrix(dc, 365, 30, arr[3]);


dc.TextOut(500, 5, _T("Результат:"));

CMatrix C1 = arr[0] + arr[1];

dc.TextOut(500, 30, _T("C1 = A + B"));

controller.PrintMatrix(dc, 500, 50, C1);


CMatrix C2 = arr[0] * arr[1];

dc.TextOut(700, 30, _T("C2 = A * B"));

controller.PrintMatrix(dc, 700, 50, C2);


CMatrix D = arr[0] * arr[2];

dc.TextOut(900, 30, _T("D = A * V1"));

controller.PrintMatrix(dc, 900, 50, D);


CMatrix q = arr[2].Transp() * arr[3];
```

```
dc.TextOut(500, 200, _T("q = V1^T * V2"));
controller.PrintMatrix(dc, 500, 250, q);

CMatrix p = arr[2].Transp() * arr[0] * arr[3];
dc.TextOut(700, 200, _T("p = V1^T * A * V2"));
controller.PrintMatrix(dc, 700, 250, p);
}
```

Задание 3.

1. Реализовать функции (не члены класса **CMatrix**)

   CMatrix VectorMult(CMatrix& V1,CMatrix& V2);

   // Вычисляет векторное произведение векторов V1 и V2

   double ScalarMult(CMatrix& V1,CMatrix& V2);

   // Вычисляет скалярное произведение векторов V1 и V2

   double ModVec(CMatrix& V);

   // Вычисляет модуль вектора V

   double CosV1V2(CMatrix& V1,CMatrix& V2);

   // Вычисляет КОСИНУС угла между векторами V1 и V2

   CMatrix SphereToCart(CMatrix& PView);

   // Преобразует сферические координаты PView точки в декартовы

   // PView(0) – r – расстояние до точки.

   // PView(1) - fi - азимут(отсчет от оси X), град.

   // PView(2) - q - угол(отсчетот оси Z), град.

// Результат: R(0)- x, R(1)- y, R(2)- z

Файлы *.h определяются в файле LibGraph.

Файлы *.cpp определяются в файле LibGraph.

Выполнить тестирование разработанных функций с использованием векторов V1 и V2.

Вычисления выполняются при выборе пункта меню **Test►Functions**

Результаты вычислений вывести в окно, используя метод **PrintMatrix (…)**.

Результаты вычислений проверить в пакете Mathcad

3.1 Реализация функций.

```
CMatrix ControllerCMatrix::VectorMult(CMatrix & V1, CMatrix & V2){
if (V1.cols() == 1 && V2.cols() == 1 && V1.rows() == V2.rows())
{
CMatrix multVector(3);
multVector(0, 0) = V1(1, 0)*V2(2, 0) - V2(1, 0)*V1(2, 0);
multVector(1, 0) = (-1)*(V1(0, 0)*V2(2, 0) - V2(0, 0)*V1(2, 0));
multVector(2, 0) = V1(0, 0)*V2(1, 0) - V2(0, 0)*V1(1, 0);
return multVector;
}
else
{
return NULL;
}
}
```

В данной функции сначала идет проверка - являются ли переданные параметры векторами, далее мы создаем экземпляр матрицы размера 3 и поочередно вычисляем их векторное произведение.

```
double ControllerCMatrix::ScalarMult(CMatrix & V1, CMatrix & V2)
{
if (V1.rows() == V2.rows() && V1.cols() == V2.cols())
{
return V1(0, 0)*V2(0, 0) + V1(1, 0)*V2(1, 0) + V1(2, 0)*V2(2, 0);
```

```
}
}
```

По аналогии вторая функция, где выполняется уже скалярное произведение.

```
double ControllerCMatrix::ModVec(CMatrix & V)
{
      return V.rows() == 3 && V.cols() == 1 ?
      sqrt((double)(V(0, 0)*V(0, 0) + V(1, 0)*V(1, 0) + V(2, 0)*V(2, 0)))
      : 0.0;
}
```

В функции ModVec вычисляется модуль, по-другому длина вектора.

```
double ControllerCMatrix::CosV1V2(CMatrix& V1, CMatrix& V2)
{
return V1.cols() == 1 && V2.cols() == 1 && V1.rows() == 3 && V2.rows() ==
3 ?
(ScalarMult(V1, V2) / (ModVec(V1)*ModVec(V2))) :
0.0;
}
```

Данная функция занимается вычислением косинуса угла между векторами, взаимодействуя при этом с вышеопределенными функциями.

```
CMatrix ControllerCMatrix::SphereToCart(CMatrix & PView)
{
CMatrix R(3);
R(0) = PView(0)*cos(PView(1))*sin(PView(2));
R(1) = PView(0)*sin(PView(1))*sin(PView(2));
R(2) = PView(0)*cos(PView(2));
return R;
}
```

Исходные векторы:

8.0	5.0	
V1 = 7.0	V2 = 3.0	
2.0	4.0	

Векторное произведение:

22.0
-22.0
-11.0

Скалярное произведение:

69.0

Модуль вектора V1:

10.8

Косинус между V1 и V2:

0.9

Преобразует сферические координаты PView точки в декартовы:

8.0	5.5
PView =7.0	4.8
2.0	-3.3

Эта функция используется для преобразования координат точек из сферических координат в декартовы.

Реализованный функционал:

LibGraph.h:

```
#ifndef LIBGRAPH
#include "CMatrix.h"
#include "stdafx.h"
#include "resource.h"
#define LIBGRAPH 1
const double pi = 3.14159;

struct CSizeD   // определяет размер (ширина, высота) прямоугольной области
{
	double cx;
	double cy;
};
//-----------------------------------------------------------------------------
struct CRectD		// определяет верхнюю левую и нижнюю правую точки прямоугольной области
{
	double left;
	double top;
	double right;
	double bottom;
	CRectD() { left = top = right = bottom = 0; };
	CRectD(double l, double t, double r, double b);
	void SetRectD(double l, double t, double r, double b);
	CSizeD SizeD();
};

// класс для операций с CMatrix
class ControllerCMatrix
{
public:
	ControllerCMatrix() = default;
	~ControllerCMatrix() = default;

#pragma region Methods
	// вывод матрицы
	void PrintMatrix(CDC& dc, int x, int y, CMatrix& M);
	// Инициализация матрицы случайными числами
	void InitMatrix(CMatrix& M);
	//Скалярное произведение векторов
	double ScalarMult(CMatrix& V1, CMatrix& V2);
	// преобразование double в CString (для вывода в окне)
```

```
	CString DoubleToString(double x);
	// модуль вектора
	double ModVec(CMatrix& V);
	// Векторное произведение
	CMatrix VectorMult(CMatrix& V1, CMatrix& V2);
	// Косинус угла между векторами
	double CosV1V2(CMatrix& V1, CMatrix& V2);
	// Преобразует сферические координаты PView  точки в декартовы
	CMatrix SphereToCart(CMatrix& PView);
	// PView(0) – r – расстояние до точки.
	// PView(1) - fi - азимут(отсчет от оси X), град.
	// PView(2) - q - угол(отсчетот оси Z), град.
	// Результат: R(0)- x, R(1)- y, R(2)- z

#pragma endregion
private:

};

#endif
```

LibGraph.cpp:

```
#include "stdafx.h"
#include "LibGraph.h"

#pragma region Controller
void ControllerCMatrix::PrintMatrix(CDC & dc, int x, int y, CMatrix & M) //
вывод матрицы/вектора
{
	for (size_t i = 0; i < M.cols(); i++)
	{
		for (size_t j = 0; j < M.rows(); j++)
		{
			dc.TextOutW(x + i * 40, y + j * 40, DoubleToString(M(j, i)));
		}
	}
}
void ControllerCMatrix::InitMatrix(CMatrix & M)
	// инициализация матрицы/вектора
{
	for (size_t i = 0; i < M.rows(); i++)
	{
		for (size_t j = 0; j < M.cols(); j++)
		{
			M(i, j) = 1 + rand() % 10; // от 1 до 10
		}
	}
```

```
}
double ControllerCMatrix::ScalarMult(CMatrix & V1, CMatrix & V2)       // ска-
лярное произведение векторов
{
      if (V1.rows() == V2.rows() && V1.cols() == V2.cols())
      {
            return V1(0, 0)*V2(0, 0) + V1(1, 0)*V2(1, 0) + V1(2, 0)*V2(2, 0);
      }
}
CString ControllerCMatrix::DoubleToString(double x)
      // перевод вещественного числа в строковое представление
{
      CString s;
      s.Format(_T("%.1f"), x);
      return s;
}
double ControllerCMatrix::ModVec(CMatrix & V)
      // модуль вектора
{
      return V.rows() == 3 && V.cols() == 1 ?
            sqrt((double)(V(0, 0)*V(0, 0) + V(1, 0)*V(1, 0) + V(2, 0)*V(2, 0)))
            : 0.0;
}
CMatrix ControllerCMatrix::VectorMult(CMatrix & V1, CMatrix & V2)      //
векторное произведение векторов
{
      if (V1.cols() == 1 && V2.cols() == 1 && V1.rows() == V2.rows())
      {
            CMatrix multVector(3);
            multVector(0, 0) = V1(1, 0)*V2(2, 0) - V2(1, 0)*V1(2, 0);
            multVector(1, 0) = (-1)*(V1(0, 0)*V2(2, 0) - V2(0, 0)*V1(2, 0));
            multVector(2, 0) = V1(0, 0)*V2(1, 0) - V2(0, 0)*V1(1, 0);
            return multVector;
      }
      else
      {
            return NULL;
      }
}
double ControllerCMatrix::CosV1V2(CMatrix& V1, CMatrix& V2)                //
косинус угла между векторами
{
      return V1.cols() == 1 && V2.cols() == 1 && V1.rows() == 3 && V2.rows() == 3
?
            (ScalarMult(V1, V2) / (ModVec(V1)*ModVec(V2))) :
            0.0;
}
CMatrix ControllerCMatrix::SphereToCart(CMatrix & PView)                   //
```

```
преобразует сферические координаты точки в декартовы
{
	CMatrix R(3);
	R(0) = PView(0)*cos(PView(1))*sin(PView(2));
	R(1) = PView(0)*sin(PView(1))*sin(PView(2));
	R(2) = PView(0)*cos(PView(2));
	return R;
}
#pragma endregion



CRectD::CRectD(double l, double t, double r, double b)
{
	left = l;
	top = t;
	right = r;
	bottom = b;
}
//-----------------------------------------------------------------------------
void CRectD::SetRectD(double l, double t, double r, double b)
{
	left = l;
	top = t;
	right = r;
	bottom = b;
}

//-----------------------------------------------------------------------------
CSizeD CRectD::SizeD()
{
	CSizeD cz;
	cz.cx = fabs(right - left);// Ширина прямоугольной области
	cz.cy = fabs(top - bottom);// Высота прямоугольной области
	return cz;
}
```